Client-side web applications : SPMS Project

Timothée RABOEUF - 74764

December 2018

This document is a report for the graded project "Stock Portfolio Management System" for the course "Development of Client-side Interactive Web Applications", at Abo Akademi University. The source code is accessible at `https://github.com/it-teaching-abo-akademi/2018-interactive-web-apps-project-Timothee-Raboeuf`, and a live version is deployed at `https://it-teaching-abo-akademi.github.io/2018-interactive-web-apps-project-Timothee-Raboeuf/index.html`.

Project structure

Even though originally based on in-browser trans(com)pilation with Babel, this project switched to a npm-based application (see commit `96b2f95`). This choice has been motivated by the facility of the integration of third-party plugins (namely the `react-chartjs-2` dependency) into the code base : the installation of a single npm module is indeed easier than its integration within Babel.

Besides React, this project relies on the Bootstrap CSS framework for its responsive rendering. React, React-DOM, Bootstrap and their respective dependencies are loaded in the *index.html* file, whose DOM tree is composed of a single `container` node. React components are then added to this root node.

App component

The main component is the `App` component : it handles the whole state of the application. Most of the data lives in the `portfolios` stateful array : each element of this array is a JS object representing a Portfolio. Each Portfolio is composed of a unique key, a name, a currency ('€' for EUR or '$' for USD), and a list of stocks. Each portfolio also stores his own conversion rate to be able to switch between both currencies.

The whole application state is stored in the `App` component, therefore this component is responsible for loading and saving this state in the local storage, and handling data changes requested by children components : all state manipulation methods are therefore class methods of this `App` component.

Finally, the `App` component also stores data from remote APIs : the USD to EUR conversion rate from fixer.io API, and stocks values from IEXtrading API. So as to

limit the amount of remote calls to these APIs (the fixer.io API having a limitation of **XXX** requests by month) and obtaining faster results by keeping recent data in memory rather than fetching it at each render, the app uses a cache system : the `App` component refreshes this data at regular interval.

On rendering, this `App` component maps each portfolio in its state to a `Portfolio` component, responsible for rendering its own data. It also passes down to each component event handlers and data.

Portfolio component

Each portfolio is responsible for rendering its own stocks, and letting the user rename it, add, select or delete stocks, or plotting its performance. Because the whole state of the application is stored in the `App` component, the Portfolio is not aware of the stateful data and has to call parent handlers (passed down from `App`) to perform actions on the data. Each portfolio is only aware of its unique key, to allow the `App` component to identificate it on a handler call.

On rendering, each portfolio maps its stocks stateful objects to a `Stock` component, responsible for displaying its own information. It also passes down to each `Stock` component event handlers and data from the `App` component.

Modal components

When renaming a portfolio or adding a new stock, the user is prompted information on a modal window. These modals are also React components, but maintain their own state rather than relying only on the root `App` component : the data they hold are indeed only temporary, and do not need to be saved or even known in the same way as regular application data. They only share an event handler with the parent app, to be able to submit the final data.

Graph modal component